BIOSTAT 527

5/24/23

Lecture 18

Lecture 4: Combining predictors – Part II

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

November, 2015

Boosting as descent in function space
- Boosted predictors are additive models
- AdaBoost is steepest descent on training set
- A statistical view of boosting

More surrogate losses, more boosting algorithms
- Why the e^{-yf} loss? other surrogate losses

Practical remarks and theoretical results
- Practicalities
- Theoretical results

Extensions of boosting
- Boosting for multiclass and ranking
- [Multiplicative updates algorithms]

Reading HTF: 15. Random Forests, 16. Ensembles of predictors, 8. Model inference and averaging, 10. Boosting, Murphy: 16.3, 16.3.1 Generalized Additive models (ignore the regularization, for "smoother" read "minimize loss function"), 16.4.1-5 [and optionally 8] Bosting, 3.2.4 Bayesian model averaging, 16.6.3, 16.6.1 Ensembles of predictors

I. AdaBoost is steepest descent on training set

$N \leftarrow n$ sample size

We will show that boosting is a form of (stochastic) gradient descent on the surrogate loss $\hat{L}_\phi$ (we already know from Part I that AdaBoost pushes $\hat{L}_\phi$ asymptotically towards 0).

Assume we want to minimize the surrogate loss $\hat{L}_\phi$ on the training set. For any finite $\mathcal{D}$, f and $b \in \mathcal{B}$ affect $\hat{L}_\phi$ only via the N-dimensional vectors of their values on $\mathcal{D}$ (which we will abusively denote by f, b)

$$\hat{L}_\varphi = \frac{1}{N} \sum_i \varphi(y^i f(x^i))$$

$$f = \begin{bmatrix} f(x^1) \\ f(x^2) \\ \dots \\ f(x^N) \end{bmatrix} \quad b = \begin{bmatrix} b(x^1) \\ b(x^2) \\ \dots \\ b(x^N) \end{bmatrix} \in \mathbb{R}^N \tag{4}$$

Thus, $\hat{L}_\phi(f)$ is a function of N variables, with partial derivatives

$$\frac{\partial \hat{L}_\phi}{\partial f(x^i)} = \frac{\partial}{\partial f(x^i)} \left[\frac{1}{N} \sum_{i=1}^{N} \phi(y^i f(x^i)) \right] = \frac{1}{N} y^i \phi'(y^i f(x^i)) = -\frac{1}{N} y^i e^{-y^i f(x^i)}, \tag{5}$$

since $\phi'(z) = -e^{-z}$. Imagine a boosting step as trying to find a change βb in f which minimizes the loss $\hat{L}_\phi(f + \beta b)$. This minimization is equivalent to maximizing the decrease in loss $\hat{L}_\phi(f) - \hat{L}_\phi(f + \beta b)$.

$$\frac{\partial \hat{L}}{\partial f(x^i)} = \frac{1}{N}\varphi'(y^i f(x^i))\cdot y^i = -\frac{y^i}{N}\varphi(y^i f(x^i))$$

↑ gradient at current f

$$\boxed{\hat{L}_\varphi = \frac{1}{N}\sum_i \varphi(y^i f(x^i))}$$

$$\varphi(z) = e^{-z}$$

$$\varphi'(z) = -e^{-z} = -\varphi$$

Gradient descent $\quad f \leftarrow f - \beta \nabla_f \hat{L}_\varphi(f)$

β: step size; $\nabla_f \hat{L}_\varphi(f)$: b direction of descent

⇓

$$\hat{L}_\varphi(f) - \hat{L}_\varphi(f+\beta b) \overset{=}{\tau}$$

$$= \nabla \hat{L}_\varphi(f)(\beta b) = \max_b \leftarrow \text{w.r.t } b$$

max: $\sum_{i=1}^{N}\left(+\frac{1}{N} y^i \underbrace{e^{-y^i f(x^i)}}_{w^i} b(x^i)\right) = +\frac{1}{N}\sum_i y^i b(x^i)\cdot w^i \iff \min_b \text{err}(b)$

$$\text{err} = \frac{1 - y^i b x^i}{2}$$

But $b = \begin{bmatrix} b(x^1) \\ \vdots \\ b(x^N) \end{bmatrix} \in \mathcal{B}$ weak classifiers

(Restriction!)

Training b on weighted $\mathcal{D}$ now

The direction of descent

The change in L_ϕ along "direction" b with step size β is approximately

$$\hat{L}_\phi(f) - \hat{L}_\phi(f+\beta b) \approx -\left(\nabla_f \hat{L}_\phi(f)\right)^T(\beta b) = \sum_i \left[\left(\frac{1}{N} y^i e^{-y^i f(x^i)}\right)(\beta b(x^i))\right] \propto \sum_i y^i b(x^i) w_i \tag{6}$$

(denoting/recalling $w_i \propto e^{-y_i f(x^i)}$).
The direction of steepest descent b is therefore the maximizer of

$$\operatorname*{argmax}_{b \in \mathcal{B}} \sum_i w_i y_i b(x^i) \tag{7}$$

where in the sum on the r.h.s we recognize the r of AdaBoost.

- If $b(x^i) = \pm 1$ values, then $1 - y_i b(x^i) = \mathbf{1}_{[i \text{ error}]}$, and maximizing (7) is the same as minimizing the weighted training error $\hat{L}_{01}^w$.
- If b takes real values, then $y_i b(x^i)$ is the margin of example i, and maximizing (7) is a natural objectiv for many training algorithm. Exercise Can you find examples of algorithms/predictors which do/don't maximize the loss in (7)?

More generally (we will use this later), the direction b maximizes

$$\sum_i y_i b(x^i)[-\phi'(y_i f(x^i))] \tag{8}$$

Finding the direction b is equivalent with step 1 of the AdaBoost algorithm, training a weak classifier on the weighted data. The resulting b can be seen as the best approximate of the gradient of L_ϕ in $\mathcal{B}$.

2. The line minimization ⇒ β^k optimal step size

Now let us do line minimization: find the optimal step size β in direction b. For this we take the derivative of $\hat{L}_\phi(f + \beta b)$ w.r.t β and set it to 0.

$$\frac{d\hat{L}_\phi(f + \beta b)}{d\beta} = \sum_i y_i b(x^i) \phi'(y_i f(x^i)) = -\sum_i y_i b(x^i) e^{-y_i f(x^i) - \beta y_i b(x^i)} \quad (9)$$

β is the (unique) root of

$$\sum_i w_i y_i b(x^i) e^{-\beta y_i b(x^i)} = 0 \quad (10)$$

If
- $b(x) \in \{-1, 1\}$ then line optimization gives β^k from AdaBoost
- $b(x) \in [-1, 1]$ then line optimization gives β^k from AdaBoost approximately
- $b(x) \in (-\infty, \infty)$ then β amounts to a rescaling of b and is redundant.

$\sum w_i = \varepsilon^k$ mistake

$$(1-\varepsilon^k) e^{-\beta} = \varepsilon^k e^{\beta}$$

$$\beta^k = \frac{1}{2} \ln \frac{1-\varepsilon^k}{\varepsilon^k}$$

Calculating β^k for binary b's

Assume $b(x) \in \{\pm 1\}$.
In this case $y^i b(x^i) = \pm 1$ and we obtain

$$\frac{d\hat{L}_\phi(f + \beta b)}{d\beta} = \sum_{i \ corr} w_i e^{-\beta} - \sum_{i \ err} w_i e^{\beta} = 0 \tag{11}$$

$$0 = (1 - \sum_{i \ err} w_i) - \underbrace{(\sum_{i \ err} w_i)}_{\varepsilon^k} e^{2\beta} \tag{12}$$

$$\beta = \frac{1}{2} \ln \frac{1 - \varepsilon^k}{\varepsilon^k} \tag{13}$$

This is the β^k coefficient of step 4 of AdaBoost

Hence, the AdaBoost algorithm can be seen as minimizing the loss $L_\phi(f)$ by steepest descent in the function space span $\mathcal{B}$.

1+2 : Boosting = grad descent in f with direction of descent restricted to $\mathcal{B}$ & Loss L_φ

The third case corresponds to the RealAdaBoost in the FHT variant, described here for completeness

Real AdaBoost Algorithm (in the FHT variant)

Assume $\mathcal{B}$ contains real-valued functions

Input M, labeled training set $\mathcal{D}$

Initialize $f = 0$

$w_i^1 = \frac{1}{N}$ weight of datapoint x^i

for $k = 1, 2, \ldots M$

"learn classifier for $\mathcal{D}$ with weights $w^k \Rightarrow b^k$"

compute new weights $w_i^{k+1} = w_i^k e^{-y^i b^k(x^i)}$ and normalize them to sum to 1

Output $f(x) = \sum_{k=1}^{M} b^k(x)$

3. A statistical view of boosting

On the distribution P_{XY}

It has been shown [Friedman et al., 1999] (FHT) that boosting can also be seen as noisy gradient descent in function space when we replace the finite training set with the true data distribution. The loss function and gradient can be given a probabilistic interpretation. This point of view is useful in two ways:

1. It shows that boosting is asymptotically minimizing a reasonable loss function, so that we can expect the performace/and algorithm behavior on finite samples to be a good predictor on its behaviour with much larger samples.
2. It is an interpretation that allows on to create a very large variety of boosting algorithms, like the LogitBost, Gentle AdaBoost and GradientBoost, presented hereafter.

Assume

- we do boosting "at the distribution level", i.e using P_{XY} instead of the empirical distribution given by $\mathcal{D}$.
- The loss function is $L_\phi(f) = E[e^{-yf(x)}]$.
  The notation $E[]$ denotes expectation w.r.t the joint P_{XY} distribution.
- learning a classifier means "find the best possible minimizer to $L_\phi(f)$" in $\mathcal{F}$

Is L_ϕ a good loss?

Proposition

Denote $p_x = P_{XY}(y = 1|x)$. *The loss* $L_\phi(f)$ *is minimized by*

$$f^*(x) \;=\; \frac{1}{2} \ln \frac{P_{XY}(y = 1|x)}{P_{XY}(y = -1|x)} \;=\; \frac{1}{2} \ln \frac{p_x}{1 - p_x}$$

And $p_x \;=\; \frac{e^{f(x)}}{e^{f(x)} + e^{-f(x)}}$ *the logistic function.*

Exercise Does the expresion of p_x look familiar? What is the connection?

Proof Since we are minimizing over all possible f's with no restrictions, we can minimize separately for every $f(x)$. Hence, let x be fixed

$$E_{P_{Y|X=x}}[e^{-yf(x)}] \;=\; P(y = 1|x)e^{-f(x)} + P(y = -1|x)e^{f(x)}$$

and the gradient is

$$\frac{\partial E[e^{-yf(x)}|x]}{\partial f(x)} \;=\; -P(y = 1|x)e^{-f(x)} + P(y = -1|x)e^{f(x)}$$

By setting this to 0 the result follows. □

In summary f^* is the Bayes optimal predictor for L_ϕ. But by the Proposition, f^* is also Bayes optimal for L_{01}. **(Good!)**

Steepest descent on $L_\phi(f)$ is (like) RealAdaboost

Proposition

The Real AdaBoost (with "learn a classifier" defined at the distribution level) fits an additive logistic regression model f by iterative descent on $L_\phi(f)$.

Proof The proof is similar to that for the training set case.
Suppose we have a current estimate $f(x)$ and seek to improve it by minimizing $L_\phi(f+b)$ over b. In the proof we assume that b is an arbitrary function, while in practice b will be chosen to best approximate the ideal f within the class $\mathcal{B}$.
Denote by $p_x = P[y=1|x]$ (the true value) and by $\hat{p}_x$ the "estimate"

$$\hat{p}_x = \frac{e^{f(x)}}{e^{f(x)} + e^{-f(x)}} \tag{14}$$

Assume again x is fixed. Then direction

wrt $P_{y|x}$: $p_x = P[y=1|x]$

$$\begin{aligned} L_\phi(f+b) &= E[e^{-yf(x)-yb(x)}] \\ &= e^{-f(x)}e^{-b(x)}p_x + (1-p_x)e^{f(x)}e^{b(x)} \end{aligned}$$

min over $b(x)$

Taking the derivative and setting it to 0 we obtain the new step:

$$b(x) = \frac{1}{2}\ln\frac{p_x e^{-f(x)}}{(1-p_x)e^{f(x)}} = \frac{1}{2}\left[\ln\frac{p_x}{1-p_x} - \ln\frac{\hat{p}_x}{1-\hat{p}_x}\right] \tag{15}$$

Note that if one could exactly obtain the b prescribed by (15) the iteration would not be necessary.

(Proof, continued)
More interesting than the exact form of b above is the optimization problem that leads to it.
Denote $w(x, y) = e^{-yf(x)}$. Then, b is the solution of

$$b = \operatorname*{argmin}_{b \in \mathcal{B}} E_{P_{XY} w(X,Y)}[e^{-Yb}] \tag{16}$$

where $P_{XY} w(X, Y)$ denotes the (unnormalized) **twisted distribution** obtained by multiplying the original data distribution with $w(x, y)$. (Of course, one may have to put some restrictions on P_{XY} and $\mathcal{B}$ in order to obtain a proper distribution.) Finally, note that the new f is $f + b$ and the new weights are $w(x, y)e^{-yb(x)}$ which finishes the proof.
Hence, the Real AdaBoost algorithm can be seen as a form of "noisy gradient" algorithm at the distribution level. (Note that the minimization in equation (16) is over both direction and scale of f.)

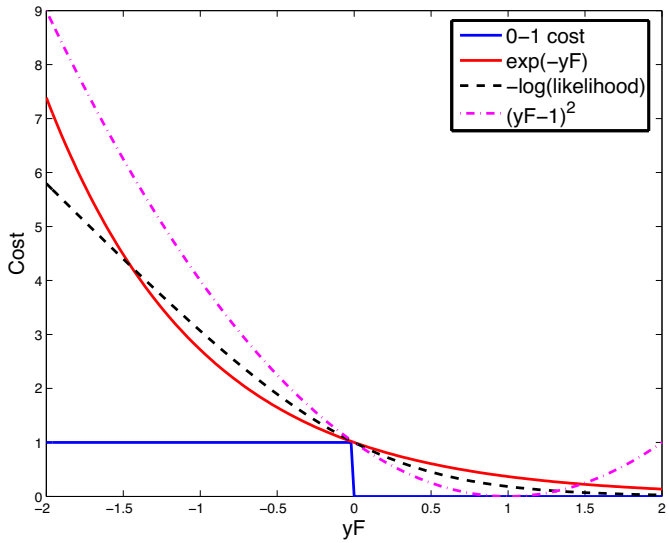
Why the e^{-yf} loss? and other L_ϕ losss

- We saw that L_ϕ is statistically motivated. Now we will see that it is computationally motivated as well.
- Recall: The "true" classification loss L_{01} is nonsmooth (has 0/no derivatives), non-convex. For training, one uses **surrogate losses**.
- Want surrogate L to have the following properties
  - $\phi(z)$ is an upper bound of the 0–1 loss
  - $\phi(z)$ is smooth (has continuous derivatives of any order if f has them); (this lets us use continuous optimization techniques to fit the classifier)
  - $\phi(z)$ is convex (this leads to global optimization, which has been recognized as beneficial in practice; it also allows to prove bounds, rates of convergence and so on)
  - $\phi(z)$ is monothone (decreasing) (thus, when $z > 0$, driving the margins to increase even if the classification is correct).

  These properties are satisfied by $L_{exp}(z) = e^{-z}$

Surrogate losses and boosting algorithms

A cornucopia of loss functions



(sometimes good to have $L(z)$ decrease for all $z < 0$, and sometimes bad – causes overfitting)

. . . and of boosting algorithms

- *GentleAdaBoost*: approx Newton on L_{exp}
- LEAST-SQUARESBOOST: many operations in closed form
- LOGITBOOST $L_{logit}(z) = \ln(1 + e^{-z})$ slower (almost linear) decrease for $z \ll 0$
- ANYBOOST, GRADIENTBOOST work with any L

GradientBoost

$L = \varphi(yf)$

GradientBoost Algorithm

Given $\mathcal{B}$ contains real-valued functions, loss L differentiable

Input M, labeled training set $\mathcal{D}$

Initialize $f^0(x) = \beta_0 = \text{argmin}_{\beta \in \mathbb{R}} \hat{L}(\beta)$

for $k = 0, 1, 2, \ldots M-1$

1. compute $r_i = -y^i \phi'(y^i f(x^i))$ } 1. direction of descent $\in \mathcal{B}$
2. fit $b^k(x)$ to outputs r_i
3. find $\beta_k = \text{argmin}_{\beta \in \mathbb{R}} \hat{L}(f^k + \beta b_k)$ (univariate optimization) 2.

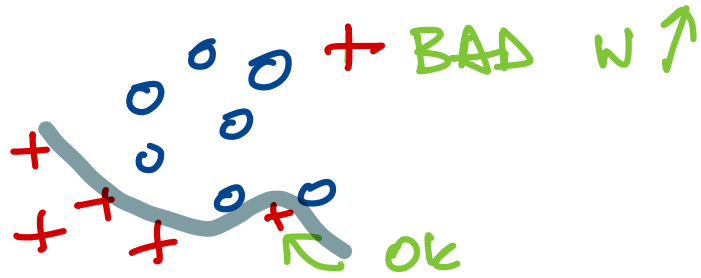
update $f^{k+1}(x) = f^k(x) + \beta^k b^k(x)$

Ouput $f^M(x)$

$M \sim N^\alpha \quad \alpha < 1$

- Can be used for either classification or regression
- Works with any L
- If L convex, step 3 is convex optimization (efficient)
- Proposed first as AnyBoost, later specialized for $\mathcal{B}$ =decision/regression trees, with other tweaks and new name GradientBoost
- When $\mathcal{B}$ =CART
  - step 3 optimizes over every leaf separately
  - depth of trees J represents maximum number of interactions in f; should not be too large ($\mathcal{B}$ must be weak)

Practical aspects



Overfitting in noise When the classes overlap much (many examples in $\mathcal{D}$ hard/impossible to classify correctly) boosting algorithms tend to focus too much on the hard examples, at the expense of overall classification accuracy. The same happens for outliers. Observe also that the loss function(s) in the previous figure, which penalize more as the margin becomes more negative.

Choice of features Often times, the base class $\mathcal{B}$ consists of function of the form $b(x) = x_j - a$, which perform a split on coordinate x_j at point $x_j = a$. They have the advantage that they can be learned and evaluated extremely fast. One can also augment the coordinate vector x with functions of the coordinates (e.g. $x \to [x_1 \ \ldots \ x_d \ x_1x_2 \ x_1x_3 \ \ldots]$) essentially creating a large set of features, which corresponds to finite but very large $\mathcal{B}$. In such a situation, the number of features d can easily be larger than M the number of b's in the final f. Thus, boosting will be implicitly performing a feature selection task.

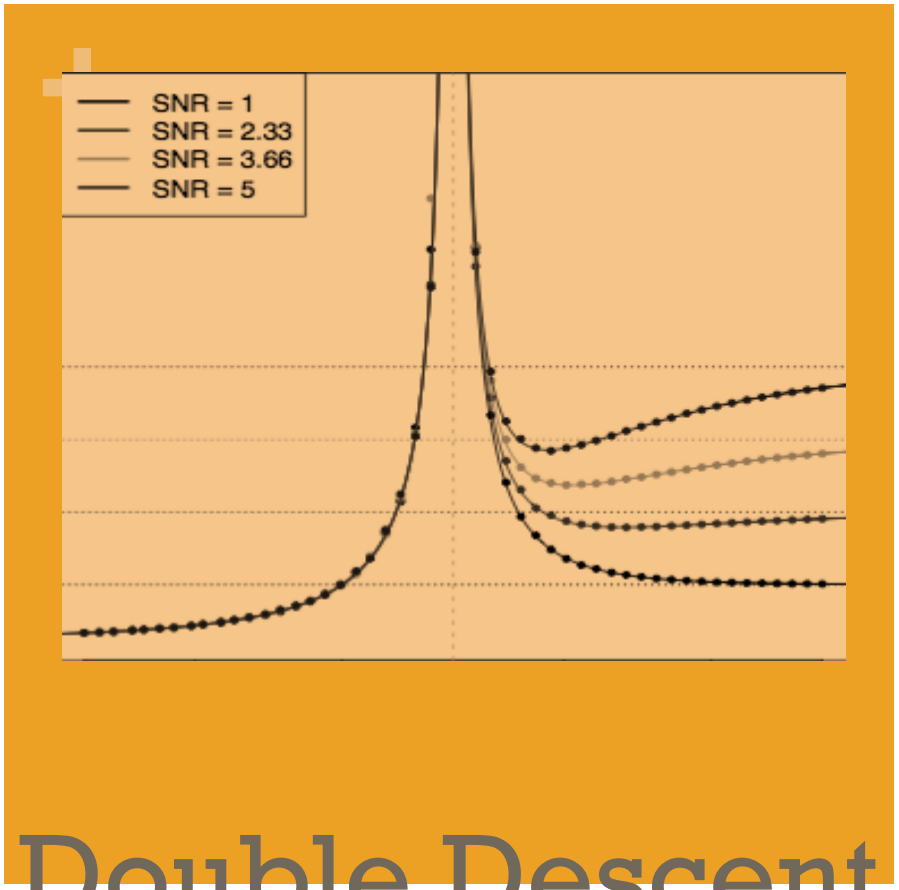
When to stop boosting?

The idea of Cross-Validation (CV) is to use an idependent sample from P_{XY}, denoted $\mathcal{D}'$ and called the **validation set** to estimate the expected loss $L_{01}(f)$. When overfitting starts, $L_{01}(f^k)$ will start increasing with k. Boosting is the stopped at the value M that minimizes $\hat{L}_{01}(b^k; \mathcal{D}')$ (denoted L_{cv} below to simplify notation)

AdaBoost with Cross-Validation

Given Training set $\mathcal{D}$ of size N, validation set $\mathcal{D}$ of size N', base classifier $\mathcal{B}$
Initialize

1. while L_{CV} decreases (but for at least 1 step)
   - do a round of boosting on $\mathcal{D}$
   - for $i' = 1 : N'$ compute $f(x^{i'}) \leftarrow f(x^{i'}) + \beta^k b^k(x^{i'})$
   - compute $L_{01}^{CV} = \frac{1}{N'} \sum_{i'} 1_{[y^{i'} f(x^{i'}) < 0]}$



Double Descent

Beyond the Bias-Variance trade-off

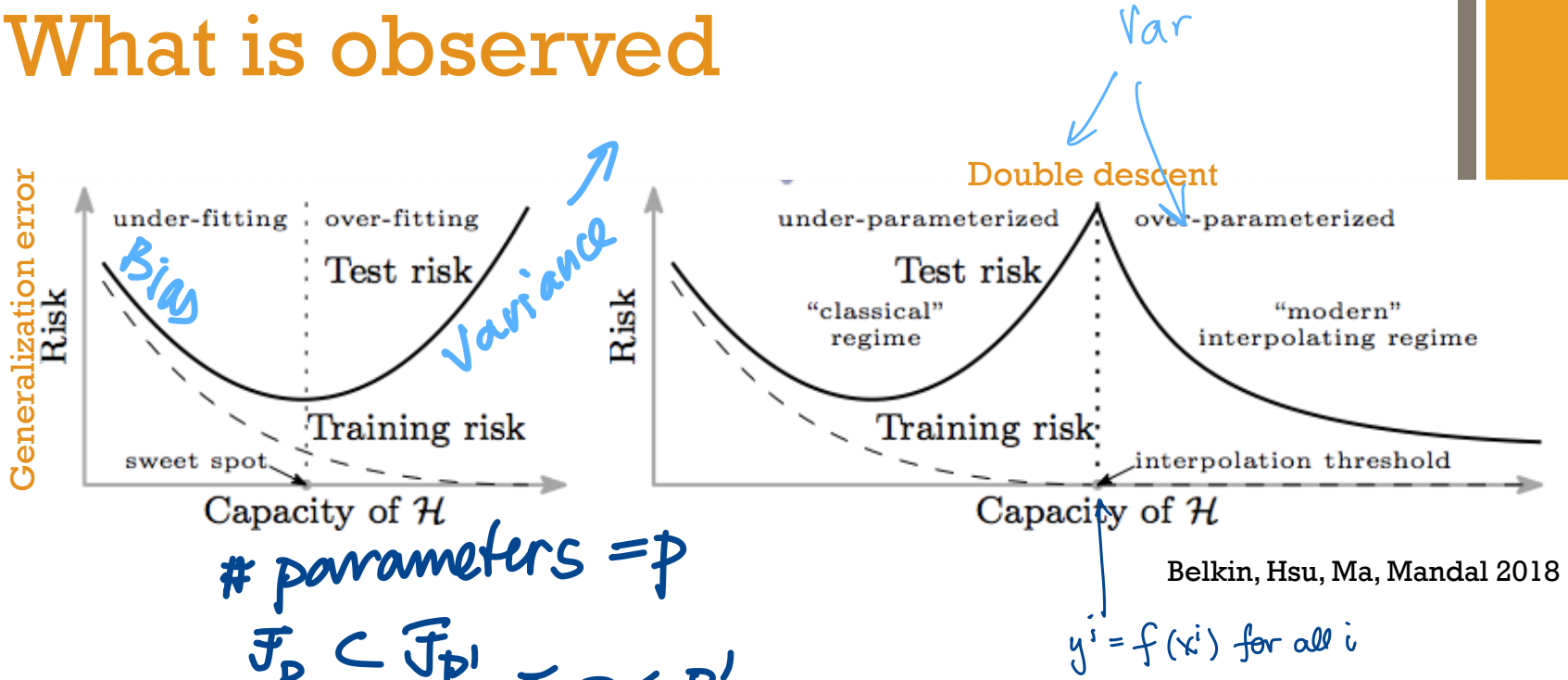
STAT 535+LPL2019

+ STAT 527

Marina Meila

University of Washington

What is observed

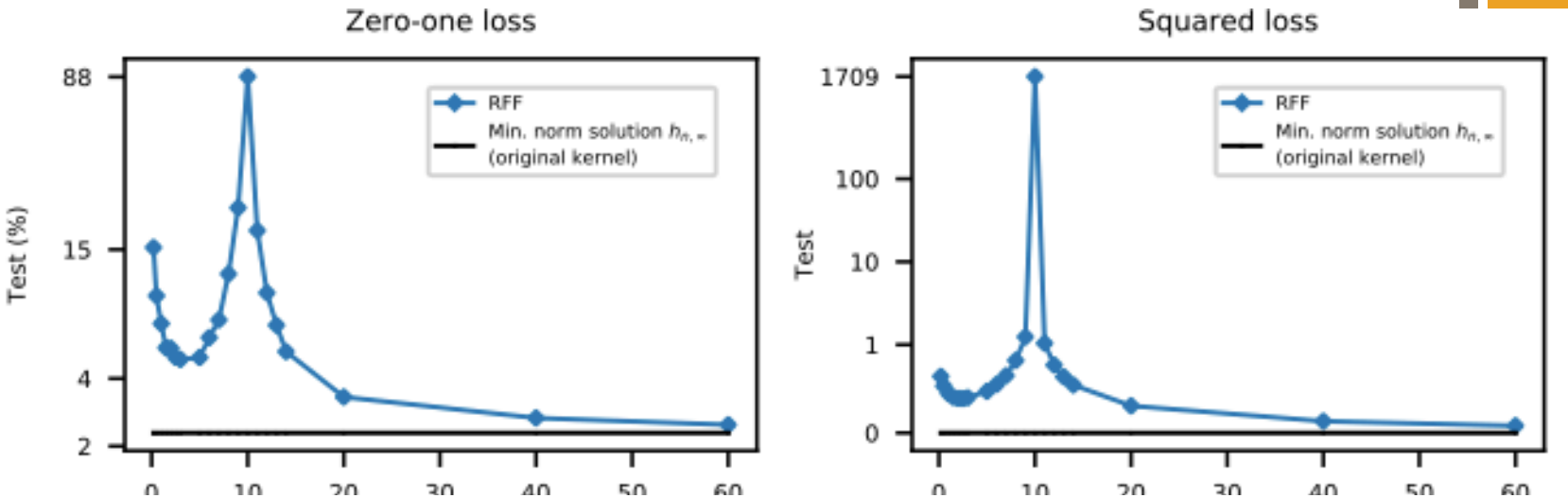


- Classical regime p < N
- Modern/Deep Learning/High dimensional regime N > n
  - Think N fixed, p increases, gamma=p/N
  - Training error = 0 (interpolation)
  - Test error decreases with p (or gamma)

$\mathcal{H} \equiv \mathcal{F}$

risk = loss

What is observed



Belkin, Hsu, Ma, Mandal 2018

- Double descent curves for the generalization error
  - Random Fourier Features (RFF)
  - ReLU 2 layer networks (with random first layer weights)
  - Random Forests, l2-Adaboost
  - Linear regression
- With and without noise

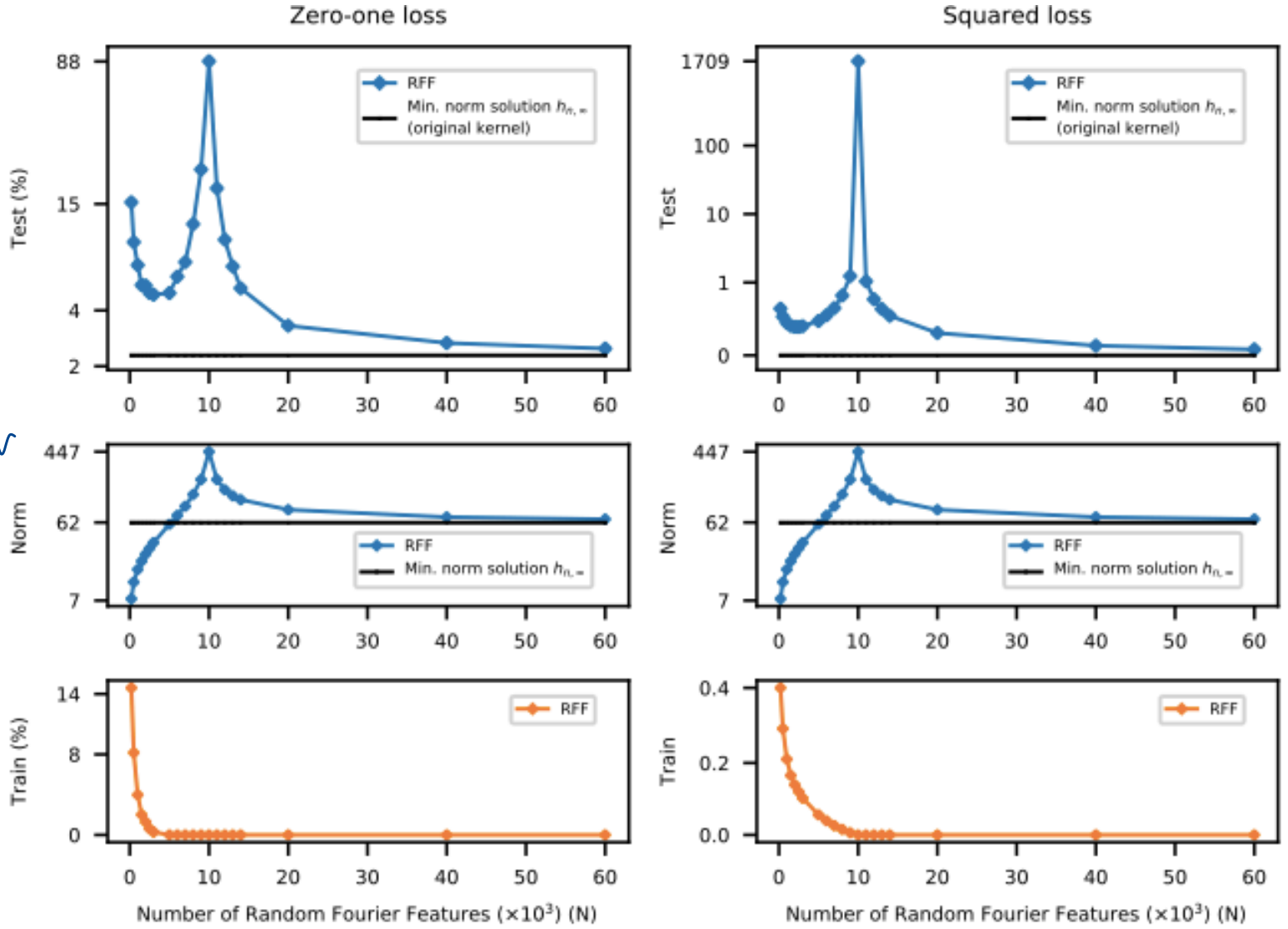
1. Thm : grad descent converges to $\hat{\theta}$ ⇒

Then $\hat{\theta} = \arg\min \|\theta\|_{RKHS}$

interpolators $\{f_\theta(x^{1:n}) = y^{1:n}\}$ in $\mathcal{F}$

$\hat{f}_\theta = \arg\min \|f_\theta\|_{RKHS}$

RFF



$\|w\|_2$

2. $\|f\|$ small $\Rightarrow$ f smooth

Theorem

If f^{true} smooth

$f^{true}(x^i) \approx f(x^i) \Rightarrow \|f - f^{true}\|_{RKHS}$ small

Theorem 1. *Fix any $h^* \in \mathcal{H}_\infty$. Let $(x_1, y_1), \ldots, (x_n, y_n)$ be independent and identically distributed random variables, where x_i is drawn uniformly at random from a compact cube[2] $\Omega \subset \mathbb{R}^d$, and $y_i = h^*(x_i)$ for all i. There exists absolute constants $A, B > 0$ such that, for any interpolating $h \in \mathcal{H}_\infty$ (i.e., $h(x_i) = y_i$ for all i), so that with high probability*

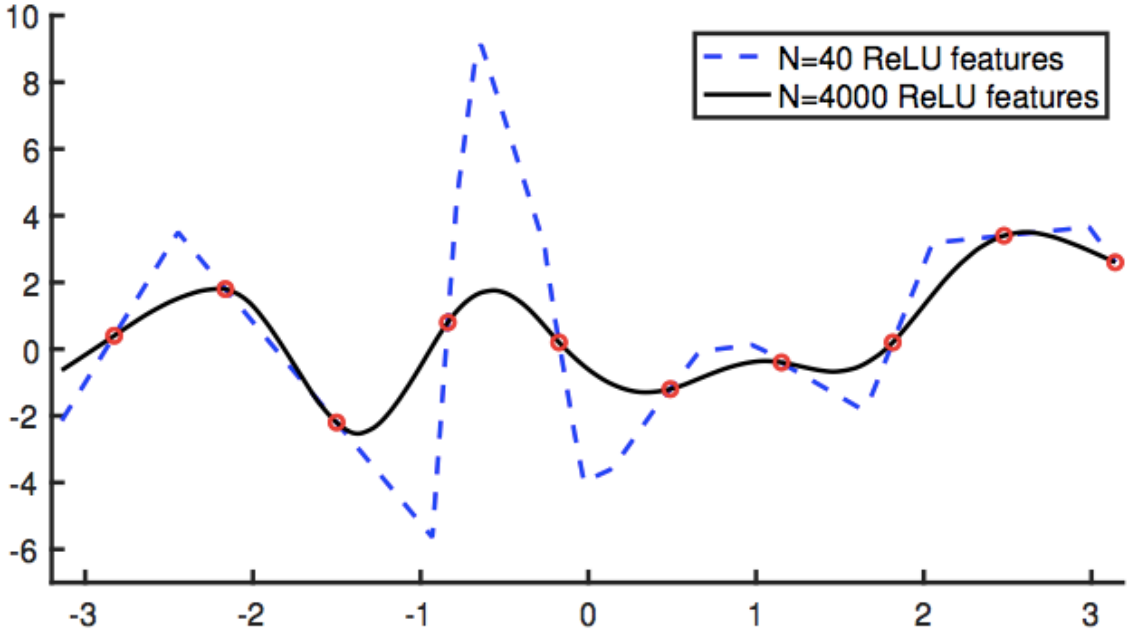
$$\sup_{x \in \Omega} |h(x) - h^*(x)| < Ae^{-B(n/\log n)^{1/d}} \left(\|h^*\|_{\mathcal{H}_\infty} + \|h\|_{\mathcal{H}_\infty}\right).$$

3. $\mathcal{F}' \subset \mathcal{F}$

$\|\hat{f}'\| \geq \|\hat{f}\|$

smoothest interpolator becomes smoother

Main intuition [Belkin et al.]



- The target function h* is (mostly) smooth
  - i.e. $||h^*||_{RKHS}$ is small
- p > N, no noise, hence h_p interpolates data
- Train to minimize $||h_p||$ subject to 0 training error
- Then $||h_p||$ will decrease with p!

Random Fourier Features (RFF)

Random Fourier features. We first consider a popular class of non-linear parametric models called *Random Fourier Features* (*RFF*) [30], which can be viewed as a class of two-layer neural networks with fixed weights in the first layer. The RFF model family $\mathcal{H}_N$ with N (complex-valued) parameters consists of functions $h\colon \mathbb{R}^d \to \mathbb{C}$ of the form

$$h(x) = \sum_{k=1}^{N} a_k \phi(x; v_k) \quad \text{where} \quad \phi(x; v) := e^{\sqrt{-1}\langle v, x \rangle},$$

and the vectors $v_1, \ldots, v_N$ are sampled independently from the standard normal distribution in $\mathbb{R}^d$. (We consider $\mathcal{H}_N$ as a class of real-valued functions with $2N$ real-valued parameters by taking real and imaginary parts separately.) Note that $\mathcal{H}_N$ is a randomized function class, but as $N \to \infty$, the function class becomes a closer and closer approximation to the Reproducing Kernel Hilbert Space (RKHS) corresponding to the Gaussian kernel, denoted by $\mathcal{H}_\infty$.

- RFF → $\mathcal{H}_{\text{infinity}}$

Boosted decision trees